

REMARKS/ARGUMENTS

Claims 1-22 are pending. Claim 16-22 are amended. Support for the amendment to claim 16 can be found in the specification on page 17, lines 1-12.

I. Objection to the Drawings

The Office Action objects to Figure 3 as not showing “file system 300” as described in the specification. Applicants have amended Figure 3 accordingly, thereby overcoming the objection.

II. 35 U.S.C. § 101: Asserted Non-Statutory Subject Matter

The Office Action rejects claim 16 as directed towards non-statutory subject matter. Applicants have amended claims 16-22 in a manner similar to the Office Action’s suggested style of amendment, only in order to expedite prosecution. In any case, these claims have been amended to recite a recordable-type medium having a computer program product, which is statutory subject matter under 35 U.S.C. § 101. Therefore, this rejection has been overcome.

III. 35 U.S.C. § 102: Asserted Anticipation

III.A. Anticipation Rejections Generally

Generally, the Office Action ignores features of the claims, or at least mischaracterizes the plain language of the claims in view of the references. The references are directed to the use of inodes. However, the Office Action appears to ignore the fact that none of the references explicitly teach, “*responsive to* space being available, storing the data in the inode,” as in claim 1 (a feature present in all of the claims). Similarly, the Office Action appears to ignore the fact that none of the references explicitly teach “*determining whether space is available in an inode* for a file in the file system,” as in claim 1 (a feature present in all of the claims).

Anticipation focuses on whether a claim reads on the product or process a prior art reference discloses, not on what the reference broadly teaches. *Kalman v. Kimberly-Clark Corp.*, 713 F.2d 760, 218 U.S.P.Q. 781 (Fed. Cir. 1983). Thus, in stating an anticipation rejection, the Office Action may not rely on the general disclosures of the references, but *must* rely only on the explicit teachings of the references. Because *none* of the references explicitly teach the recited features of the claims, in the manner claimed, none of the references anticipate the claims.

Similarly, none of the references inherently teach the features of claim 1 described above. The fact that a certain result or characteristic *may* occur or be present in the prior art is not sufficient to establish the inherency of that result or characteristic. *In re Rijckaert*, 9 F.3d 1531, 1534, 28 USPQ2d

1955, 1957 (Fed. Cir. 1993); *In re Oelrich*, 666 F.2d 578, 581-82, 212 USPQ 323, 326 (CCPA 1981). "To establish inherency, the extrinsic evidence 'must make clear that the missing descriptive matter is *necessarily* present in the thing described in the reference, and that it would be so recognized by persons of ordinary skill. *Inherency, however, may not be established by probabilities or possibilities.* The mere fact that a certain thing may result from a given set of circumstances is not sufficient.'" *In re Robertson*, 169 F.3d 743, 745, 49 USPQ2d 1949, 1950-51 (Fed. Cir. 1999).

In the case at hand, the features of "determining" whether space is available and, "responsive to spacing being available, storing the data in the inode" are not *necessarily present* in the references. The determining feature is not necessarily present because storage of data in an inode can be performed without determining whether space is available. An error condition may occur if space is not available, but if space is available then the storage will occur. Even if the "determining" feature was *probably* present in the references, this fact would not be sufficient to establish an inherent disclosure under the standards of *In re Robertson*.

Additionally, the feature of "responsive to space being available, storing..." is not *necessarily present* in the references. The "responsive to" feature is not necessarily present because data may or may not be stored in an inode *responsive to* the space being available. For example, space may be available, but data not stored in the inode. Even if the "responsive to" feature was *probably* present in the references, this fact would not be sufficient to establish an inherent disclosure under the standards of *In re Robertson*.

Because the "determining" and "responsive to" features of claim 1 are not *necessarily present* in the references, the Office Action cannot assert that the references anticipate claim 1. Therefore, no anticipation rejection can be stated against claim 1 or any other claim.

In view of these facts, Applicants now rebut the specific rejections made by the Office Action. As shown below, none of the references *explicitly teach* all of the features of claim 1. Therefore, under the standards of *In re Bond*, *Kalman v. Kimberly Clark Co.*, and *In re Robertson*, none of the references anticipate claim 1 or any other claim.

III.B. Rejection in view of *Shinkai*

The Office Action rejects claims 1, 8, 9, and 16 under 35 U.S.C. § 102 as anticipated by *Shinkai, Method and Apparatus for Managing File, Computer Product, and File System*, U.S. Patent Application Publication 2005/0234867 (October 20, 2005) (hereinafter "*Shinkai*"). This rejection is respectfully traversed. The Office Action states that:

Shinkai discloses in figures 2, 6-9, a method in a data processing system for storing data in a file system (Abstract), the method comprising determining whether space is available in an inode for a file in the file system (Paragraph

[0056], [0076]); and responsive to space being available, storing the data in the inode (paragraph [0076], [0079]).

Office action of July 16, 2007, p. 5.

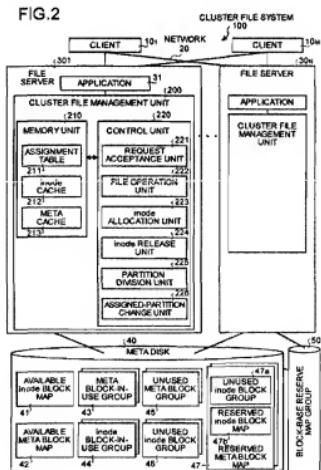
A prior art reference anticipates the claimed invention under 35 U.S.C. § 102 only if every element of a claimed invention is identically shown in that single reference, arranged as they are in the claims. *In re Bond*, 910 F.2d 831, 832, 15 U.S.P.Q.2d 1566, 1567 (Fed. Cir. 1990). All limitations of the claimed invention must be considered when determining patentability. *In re Lowry*, 32 F.3d 1579, 1582, 32 U.S.P.Q.2d 1031, 1034 (Fed. Cir. 1994). Anticipation focuses on whether a claim reads on the product or process a prior art reference discloses, not on what the reference broadly teaches. *Kalman v. Kimberly-Clark Corp.*, 713 F.2d 760, 218 U.S.P.Q. 781 (Fed. Cir. 1983). In this case, each and every feature of the presently claimed invention is not identically shown in the cited reference, arranged as they are in the claims.

Applicants first address the rejection of claim 1. Claim 1 is as follows:

1. A method in a data processing system for storing data in a file system, the method comprising:
 - determining whether space is available in an inode for a file in the file system; and
 - responsive to space being available, storing the data in the inode.

Shinkai does not anticipate claim 1 because *Shinkai* does not teach all of the features of claim 1.

The Office Action asserts otherwise, citing the following portions of *Shinkai*:



[0033] FIG. 2 is a functional block diagram of a system configuration of a cluster file system 100 according to the embodiment. The cluster file system 100 includes clients 10.sub.1 to 10.sub.M, file servers 30.sub.1 to 30.sub.N, a Meta disk 40, and a data disk 50. The clients 10.sub.1 to 10.sub.M and the file servers 30.sub.1 to 30.sub.N are connected to one another through a network 20, and the file servers 30.sub.1 to 30.sub.N share the Meta disk 40 and the data disk 50.

Shinkai, paragraph 0033.

Figure 2 of *Shinkai* shows a system configuration for a cluster file system. Figure 2 of *Shinkai* references the term “inode” many times, such as in reference numerals 212, 223, 224, 41, 47a, 47b, 44, and 46. However, nothing in Figure 2 of *Shinkai* teaches the claimed features of “determining” and “responsive to” in the manner recited in claim 1.

Nevertheless, the Office Action also cites to the flowcharts in Figures 6-9 of *Shinkai*. Figure 6 is a flowchart of a process procedure for a request acceptance unit shown in Figure 2. Figure 7 is a flowchart of a process procedure for a file operation unit shown in Figure 2. Figure 8 is a flowchart of a process procedure for an inode allocation unit shown in Figure 2. Figure 9 is a flowchart of a process procedure for an inode release unit shown in Figure 2.

However, in none of these flowcharts does *Shinkai* teach the claimed features of “determining” and “responsive to” in the manner recited in claim 1. Assuming, *arguendo*, that *Shinkai* discloses storing data in an inode block, the claimed features of “determining” and “responsive to,” in the manner recited in claim 1, are still absent from *Shinkai*.

Nevertheless, the Office Action also refers to the following portion of *Shinkai*:

[0056] The inode allocation unit 223 is a function unit that acquires an inode block required when a file or directory is created. The file server that manages the partition with the partition number of 0 acquires an available inode block using the available inode block map 41, and a file server that manages a partition with any partition number other than 0 acquires an available inode block using the reserved inode block map 47a.

Shinkai, paragraph 0056.

This portion of *Shinkai* teaches that the inode allocation unit acquires an inode block required when a file or directory is created. The file server acquires an available block.

However, again, *Shinkai* does not teach the claimed features of “determining” and “responsive to” in the manner recited in claim 1. Assuming, *arguendo*, that *Shinkai* discloses storing data in an inode block, the claimed features of “determining” and “responsive to,” in the manner recited in claim 1, are still absent from *Shinkai*.

Nevertheless, the Office Action also refers to the following portion of *Shinkai*:

[0076] If the partition number of an inode block to be allocated is not 0, the inode allocation unit 223 acquires an available inode number using the reserved inode block map 47a corresponding to the partition number (step S805), allocates the inode block (step S806), and updates the reserved inode block map 47a (step S807). The inode allocation unit 223 checks whether the number of available inode blocks becomes a predetermined value or less (step S808). If it is not the predetermined value or less, the process is ended. On the other hand, if the number of available inode blocks becomes the predetermined value or less, the inode allocation unit 223 makes an inode reserve request (step S809), and updates the reserved inode block map 47a (step S810).

Shinkai, paragraph 0076.

The cited portion of *Shinkai* teaches that if the partition number of an inode block to be allocated is not 0, then the inode allocation unit acquires an available inode number. The inode allocation unit then allocates the inode block and updates the reserved inode block map. If the number of available inode blocks falls below a predetermined value, the inode allocation unit makes an inode reserve request and updates the reserved inode block map.

However, *Shinkai* still does not teach “determining whether space is available in an inode for a file in the file system,” as required by claim 1. Instead, *Shinkai* teaches *determining whether inodes are available*. Additionally, *Shinkai* does not teach, “responsive to space being available, storing the data in the inode,” as required by claim 1. Instead, *Shinkai* teaches *responsive to there being insufficient inodes, requesting more inodes*. These features are entirely distinct from each other because *Shinkai* teaches manipulating the inodes themselves, not determining whether space is available in an inode and then responsive to space being available storing the data in the inode, as required by claim 1.

Nevertheless, the Office Action also refers to the following portion of *Shinkai*:

[0079] If the number of available inode blocks is the predetermined value or more, the inode release unit 224 notifies a file server that manages the partition 0 of releasing of the available inode block reserved (step S905), and updates the reserved inode block map 47a (step S906). In this case, the file server that manages the partition 0 updates the available inode block map 41, performs synchronous writing in the inodes 320, and requests the whole file servers to invalidate the inode cache.

Shinkai, paragraph 0079.

This portion of *Shinkai* teaches that if the number of inode blocks is equal to or greater than the predetermined value, then the inode release unit causes reserved inode blocks to be released. The inode release unit then updates the reserved inode block map accordingly. In this case, the file server updates the inode block map, performs synchronous writing in the inodes, and requests the file servers to invalidate the inode cache.

However, *Shinkai* still does not teach “determining whether space is available in an inode for a file in the file system,” as required by claim 1. Instead, *Shinkai* teaches manipulation of the inodes themselves. Additionally, even assuming, *arguendo*, that “synchronous writing in the inodes” as in *Shinkai* could be considered storing data in an inode as claimed, such writing is not determining whether space is available in the inode, as required by claim 1. Additionally, such writing does not occur *responsive to* space being available. Therefore, *Shinkai* also does not teach, “responsive to space being available, storing the data in the inode,” as in claim 1.

As shown above, none of the portions referred-to by the Office Action teach all of the features of claim 1. Additionally, no other portion of *Shinkai* teaches the features of claim 1. Accordingly, under the

standards of *In re Bond*, *Shinkai* does not anticipate claim 1.

Claims 8, 9, and 16 contain features similar to those presented in claim 1. Hence, for the reasons presented above, *Shinkai* also does not anticipate these claims.

III.C. Rejection in view of *Jiang*

The Office Action rejects claims 1, 8, 9, and 16 under 35 U.S.C. § 102 as anticipated by *Jiang*, et al., Delegation of Metadata Management in a Storage System by Leasing of Free File System Blocks and I-Nodes from a File System Owner, U.S. Patent Application Publication 2003/0191745 (October 9, 2003) (hereinafter “*Jiang*”). This rejection is respectfully traversed. The Office Action states that:

Jiang discloses in figures 5-7, 11, a method in a data processing system for storing data in a file system (Abstract), the method comprising determining whether space is available in an inode for a file in the file system (Paragraph [0011], [0064]); and responsive to space being available, storing the data in the inode (paragraph [0011], [0064]).

Office action of July 16, 2007, pp. 5-6.

Applicants first address the rejection of claim 1. For convenience, Applicants reproduce claim 1 again below:

1. A method in a data processing system for storing data in a file system, the method comprising:
 - determining whether space is available in an inode for a file in the file system; and
 - responsive to space being available, storing the data in the inode.

Jiang does not anticipate claim 1 because *Jiang* does not teach all of the features of claim 1. The Office Action asserts otherwise, citing the following

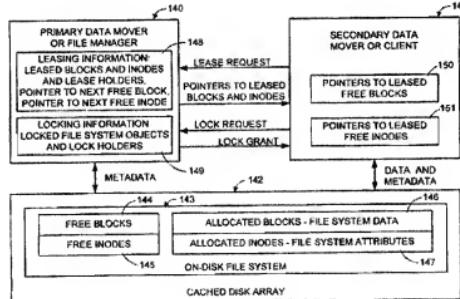


Fig. 5

respect to the file system. The on-disk file system 143 includes free blocks 144,

[0048] FIG. 5 shows the management of metadata by a primary data mover or file manager 140 that owns a file system 143 in storage of a cached disk array 142, and a data mover or client 141 that is secondary with

free inodes 145, allocated blocks 146, and allocated inodes 147. The allocated blocks 146 may contain valid file or directory data, and the allocated inodes 147 may contain valid file or directory attributes.

Jiang, paragraph 0048.

Figure 5 of *Jiang* shows the operation of *Jiang*'s primary data mover or file manager. A data mover manages files. The on-disk file system 143 includes allocated inodes, which may contain valid file attributes or directory attributes.

However, nothing in figure 2 of *Shinkai* teaches the claimed features of "determining" and "responsive to" in the manner recited in claim 1. The disclosure simply does not exist.

The Office Action also refers to the flowcharts in figures 6, 7, and 11 of *Jiang*. Figures 6 and 7 together are a flow chart showing a method of delegating metadata management from the file manager or primary data mover to the client or secondary data mover in figure 5. Figure 11 is a flow chart showing specific steps for performing a file deletion operation in accordance with the method of figure 7.

However, these figures are related to manipulation of sets of inodes themselves. As with *Shinkai*, the Office Action appears to confuse the manipulation of inodes with the requirements of claim 1. In none of these flowcharts does *Jiang* teach the claimed features of "determining" and "responsive to" in the manner recited in claim 1. Assuming, *arguendo*, that *Jiang* discloses storing data in an inode block, the claimed features of "determining" and "responsive to," in the manner recited in claim 1, are still absent from *Jiang*.

Additionally, figure 7 of *Jiang* teaches that the secondary data mover uses the inode number of the object to perform an append, truncate, create, or delete operation upon a file without assistance from the primary data mover. *Jiang*, figure 7, reference numeral 168. However, these figures do not teach actually storing data in the inodes, as required by claim 1. Therefore, none of these figures teach any of the "determining," "responsive to," and "storing the data in the inode" features recited in claim 1.

Nevertheless, the Office Action also refers to the following portion of *Jiang*:

Metadata management in a file server or storage network is delegated from a primary data processor to a secondary data processor in order to reduce data traffic between the primary data processor and the secondary data processor. The primary data processor retains responsibility for managing locks upon objects in the file system that it owns, and also retains responsibility for allocation of free blocks and inodes of the file system. By leasing free blocks and inodes to the secondary and granting locks to the secondary, the secondary can perform the other metadata management tasks such as appending blocks to a file, truncating a file, creating a file, and deleting a file.

Jiang, Abstract.

Jiang's abstract teaches that *Jiang* is directed to a metadata management system. In particular, *Jiang* is concerned with the relationship between a primary processor and a secondary processor such that data traffic is reduced between them. *Jiang* teaches allocation of free inodes using the primary processor.

However, *Jiang*'s abstract is related to manipulation of sets of inodes themselves. As with *Shinkai*, the Office Action appears to confuse the manipulation of inodes with the requirements of claim 1. *Jiang*'s abstract does not teach the claimed features of "determining" and "responsive to" in the manner recited in claim 1. Assuming, *arguendo*, that *Jiang* discloses storing data in an inode block, the claimed features of "determining" and "responsive to," in the manner recited in claim 1, are still absent from *Jiang*.

However, this portion of *Jiang* also does not teach storing data in an inode, as required by claim 1. Instead, the inodes are used to refer to files. Thus, again, the cited portion of *Jiang* does not teach all of the features of claim 1.

Nevertheless, the Office Action also refers to the following portion of *Jiang*:

[0011] In accordance with another aspect, the invention provides a method of operating a primary data processor and a secondary data processor for access to a file system in data storage. The method includes the primary data processor managing locks upon objects in the file system, and managing allocation of free blocks and free inodes (i.e., index nodes) of the file system. The method further includes the secondary data processor creating a new file of the file system by the secondary data processor obtaining an allocation of a free inode and at least one free block from the primary data processor, the secondary data processor writing file attributes to the free inode and linking the free block to the free inode, the secondary data processor obtaining, from the primary data processor, a lock on a directory of the file system to contain an entry for the new file, and the secondary data processor inserting the entry for the new file into the directory.

Jiang, paragraph 0011.

This portion of *Jiang* teaches obtaining free inodes and then writing file *attributes* to free inodes. *Jiang* also teaches the relationship between the primary and secondary processor.

However, this portion of *Jiang* does not teach the claimed features of "determining" and "responsive to" in the manner recited in claim 1. Assuming, *arguendo*, that *Jiang* discloses storing data in an inode block, the claimed features of "determining" and "responsive to," in the manner recited in claim 1, are still absent from *Jiang*.

However, this portion of *Jiang* also does not teach storing data in an inode, as required by claim 1. Instead, *attributes of a file* are written to an inode. This teaching is not equivalent to "storing the data," as required by claim 1. Thus, again, the cited portion of *Jiang* does not teach all of the features of claim 1.

Nevertheless, the Office Action also refers to the following portion of *Jiang*:

[0064] In view of the above, there has been described a method of delegation of metadata management in a file server or storage network from a primary data processor to a secondary data processor in order to reduce data traffic between the primary data processor and the secondary data processor. The primary data processor retains responsibility for managing locks upon objects in the file system that it owns, and also retains responsibility for allocation of free blocks and inodes of the file system. By leasing free blocks and inodes to the secondary and granting locks to the secondary, the secondary can perform the other metadata management tasks such as appending blocks to a file, truncating a file, creating a file, and deleting a file.

Jiang, paragraph 0064.

This portion of *Jiang* summarizes *Jiang*'s teachings. *Jiang* states that he has taught a method of delegating metadata management in a file server from a primary processor to a secondary processor in order to reduce data traffic between them. Again, *Jiang* only teaches obtaining free inodes and then writing file *attributes* to free inodes, not writing data to the inodes in the claimed manner, not determining whether space is available *in an inode*, and not storing data “*responsive to*” space being available, as recited in claim 1.

As shown above, none of the portions referred-to by the Office Action teach all of the features of claim 1. Additionally, no other portion of *Jiang* teaches the features of claim 1. Accordingly, under the standards of *In re Bond*, *Jiang* does not anticipate claim 1.

Claims 8, 9, and 16 contain features similar to those presented in claim 1. Hence, for the reasons presented above, *Jiang* also does not anticipate these claims.

III.C Rejection in view of *Doucette*

The Office Action rejects claims 1, 8, 9, and 16 under 35 U.S.C. § 102 as anticipated by *Doucette, et al., Space Allocation in a Write Anywhere File System*, U.S. Patent Application Publication 2004/0139273 (July 15, 2004) (hereinafter “*Doucette*”). This rejection is respectfully traversed. The Office Action states that:

Doucette discloses in figures 2-3, a method in a data processing system for storing data in a file system (Abstract), the method comprising determining whether space is available in an inode for a file in the file system (Paragraph [0073]); and responsive to space being available, storing the data in the inode (paragraph [0074], [0075]).

Applicants first address the rejection of claim 1. For convenience, Applicants reproduce claim 1 again below:

1. A method in a data processing system for storing data in a file system, the method comprising:

determining whether space is available in an inode for a file in the file system; and
responsive to space being available, storing the data in the inode.

Doucette does not anticipate claim 1 because *Doucette* does not teach all of the features of claim 1.

1. The Office Action asserts otherwise, citing the following portions of *Doucette*:

The Office Action first refers to the following portion of *Doucette*:

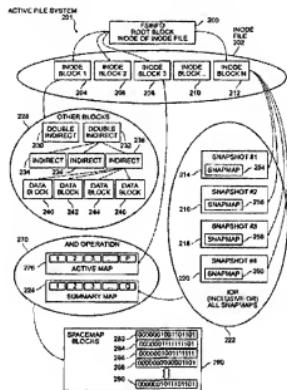


FIG. 2

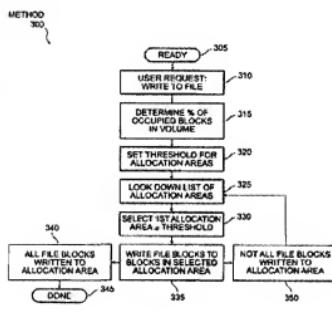


FIG. 3

Figure 2 of *Doucette* is a block diagram of a system for space allocation on disk storage. Figure 3 is a flowchart of a method for using the system of figure 2 (and also for using the system of figure 1). Figure 2 shows how inodes contain information *about* files in a file system. Figure 3 teaches writing file blocks to blocks in selected allocation areas, but is devoid of disclosure regarding the writing of data to inodes.

However, nothing in figure 2 of *Shinkai* teaches the claimed features of “determining whether space is available *in the inode*” and “responsive to space being available, storing the data in the inode” in the manner recited in claim 1. The disclosure simply does not exist.

Nevertheless, the Office Action also refers to the following portion of *Doucette*:

The invention provides a method and system for improving data access of a reliable file system. In a first aspect of the invention, the file system determines the relative vacancy of a collection of storage blocks, herein called an “allocation area”. This is accomplished by recording an array of binary numbers. Each binary number in the array describes the vacancy of a collection of storage blocks. The file system examines these binary numbers when attempting to record file blocks in relatively contiguous areas on a storage medium, such as a hard disk. When a request to write to disk occurs, the system determines the average vacancy of all the allocation areas and queries the allocation areas for

individual vacancy rates such as sequentially. The system preferably writes file blocks to the allocation areas that are above a threshold related to the average storage block vacancy of the file system. If the file in the request to write is larger than the selected allocation area, the next allocation area above the threshold is preferably used to write the remaining blocks of the file.

Doucette, Abstract.

This portion of *Doucette* teaches that a file system can be used to determine the relative vacancy of a collection of storage blocks called allocation areas. An array of binary numbers describes the vacancy of the allocation areas. When attempting to record file blocks, the file system examines the binary numbers and determines the average vacancy of all allocation areas, and queries the allocation areas for individual vacancy rates. The file system then writes file blocks to the allocation areas that are above a threshold related to the average block vacancy of the file system.

However, the cited portion of *Doucette* does not teach, “*determining whether space is available in an inode for a file* in the file system.” Such disclosure is simply absent from *Doucette*. Therefore, *Doucette* does not teach this claimed feature.

Similarly, *Doucette* does not teach, “*responsive to space being available, storing the data in the inode*.” The cited portion of *Doucette* does not teach storing the data in the inode. Additionally, the storage that does take place is not *responsive to space being available*, as claimed; but rather, is responsive to a request to write to available space. A stark difference exists between these two functions. Therefore, *Doucette* does not teach this claimed feature.

Nevertheless, the Office Action also refers to the following portion of *Doucette*:

[0073] At a step 315, the file system determines the percent of occupied blocks in the entire file system by interrogating information stored in the file system information block.

[0074] At a step 320, a threshold for allocation areas is set based on the relative allocation of all the blocks. For example, if 60% of the blocks are available for writing, then a threshold of 55% free could be picked by the file system.

[0075] At a step 325, the file system searches linearly through allocation areas looking up spacemap entries from spacemap blocks corresponding to that allocation area, looking for the first allocation area whose free space is a number representing a value greater than or equal to the threshold, in this case 55% free.

Doucette, paragraphs 0073-0075.

This portion of *Doucette* teaches determining the percent of occupied blocks in the file system, setting a threshold for allocation areas based on the allocation of all blocks, and then searching through allocation areas while looking up space map entries corresponding to the allocation areas in order to find an allocation area that is equal to or above the threshold.

However, the cited portion of *Doucette* does not teach, “*determining whether space is available in an inode for a file* in the file system.” Such disclosure is simply absent from *Doucette*. Therefore, *Doucette* does not teach this claimed feature.

Similarly, *Doucette* does not teach, “*responsive to space being available, storing the data in the inode*.” The cited portion of *Doucette* does not teach storing the data in the inode. Additionally, the storage that does take place is not *responsive to space being available*, as claimed; but rather, is responsive to a request to write to available space. A stark difference exists between these two functions. Therefore, *Doucette* does not teach this claimed feature.

Generally, *Doucette* defines inodes as follows:

[0027] Inode--In general, the term "inode" refers to data structures that include information about files in Unix and other file systems. Each file has an inode and is identified by an inode number (i-number) in the file system where it resides. Inodes provide important information on files such as user and group ownership, access mode (read, write, execute permissions) and type. An inode points to the inode file blocks.

Doucette, paragraph 0027.

Doucette specifically states that inodes refer to data structures that include information *about* files. *Doucette* does not store the files themselves. Thus, *Doucette* contains disclosure that *explicitly contradicts* the Office Action’s assertion that *Doucette* teaches “storing the data,” as required by claim 1. Accordingly, *Doucette* does not teach the other features of claim 1, as shown above.

As shown above, none of the portions referred-to by the Office Action teach all of the features of claim 1. Additionally, no other portion of *Doucette* teaches the features of claim 1. Accordingly, under the standards of *In re Bond*, *Doucette* does not anticipate claim 1.

Claims 8, 9, and 16 contain features similar to those presented in claim 1. Hence, for the reasons presented above, *Doucette* also does not anticipate these claims.

III.D Rejection in view of *Crow*

The Office Action rejected claims 1-22 as anticipated by *Crow et al., Versatile Indirection in an Extent Based File System*, U.S. Patent Application Publication 2004/0254907 (December 16, 2004) (hereinafter “*Crow*”). This rejection is respectfully traversed.

III.E.1. Claims 1, 8, 9, and 16

III.E.1.i. Response to Rejection

Claim 1 is a representative claim of this grouping of claims. Claim 1 is as follows:

1. A method in a data processing system for storing data in a file system, the method comprising:

determining whether space is available in an inode for a file in the file system; and responsive to space being available, storing the data in the inode.

The Office Action rejects claim 1 as anticipated by *Crow*. In regards to claim 1, the Office Action asserts that:

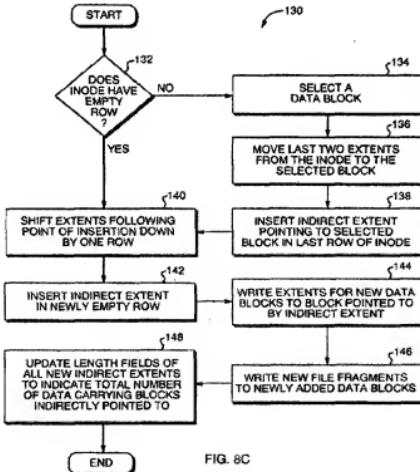
Crow discloses in figure 8C, 9-11, a method in a data processing system for storing data in a file system, the method comprising determining whether space is available in an inode for a file in the file system (132); and responsive to space being available, storing the data in the inode (142).

Office action of July 16, 2007, p 6.

Applicants thoroughly rebutted the Office Action's assertions that *Crow* anticipates claim 1 in the previously filed appeal brief. Applicants reproduce below the facts presented in the appeal brief.

Although the rejection in this office action is presented slightly differently, the Office Action's assertions remain largely the same, as the Office Action still refers to reference numerals 132 and 142 in *Crow*.

As proved in the appeal brief, *Crow* does not teach the feature of, "determining whether space is available in an inode *for a file in the file system* and responsive to space being available, storing the *data* in the inode" as recited in claim 1. The Office Action asserts otherwise, citing *Crow*'s figure 8C, reproduced below:



According to figure 8C, in step 132, "the operating system first determines whether at least one empty row remains for writing a new *extent* to the file's inode." *Crow*, paragraph 41 (emphasis added).

In step 146, the new *file fragments are stored in newly added data block, not in the inode*. The inode stores the extent, which contains a pointer that indicates both the logical volume and a physical offset of the data block. *Crow*, paragraphs 33-34. Therefore, figure 8C does not teach the features of claim 1 because figure 8C determines whether there is space in the inode for an “extent” and not for a *file in the file system* as recited in claim 1. Furthermore, figure 8C discloses storing the data in a newly added data block instead of storing the data in the inode as recited in claim 1.

Additionally, in a prior office action, the Office Action cites to the following portions of *Crow* as disclosing the features of claim 1:

[0048] Later, a request from a software application for more *data blocks for the file* is received by the operating system (step 157). In response to the request, the operating system determines whether the region contiguous to the physical location of the previous segment of the file has more available data blocks (step 158). If region has more available blocks, the operating system allocates a new string of blocks immediately following the physical location previous segment, i.e., contiguous with the previous segment (step 160). Then, the operating system increases the value of the length stored in the length field of the previous extent for the region by the number of blocks in the new string (step 161). If no blocks contiguous to the previous segment are available, the operating system again searches for a logical volume with a larger than average contiguous region of available data blocks (step 162). The newly found logical volume may be a different logical volume. Thus, the new string of data blocks may be allocated to the file from a different logical volume.

[0053] The operating system writes the binary value to the third entry 176 to indicate storage of a data file when the associated inode is first created. Then, the operating system uses the inode to store the associated data file. When the size of the data file surpasses the limited space available in the inode, the operating system converts the inode to an inode for storage of lists of extents.

Crow, paragraphs 48 and 53.

Neither the cited portion nor any other portion of *Crow* teaches the feature of, “determining whether space is available in an inode for a *file in the file system* and responsive to space being available, storing the *data* in the inode,” as recited in claim 1. Paragraph 48 only discusses an application requesting more data blocks for a file. If there are more data blocks contiguous to the physical location of the previous segment of the file, then the operating system allocates a new string of blocks immediately following the physical location of the previous segment. If no blocks contiguous to the previous segment are available, the operating system again searches for a logical volume with a larger than average contiguous region of available data blocks (step 162).

However, nothing in paragraph 48 is relevant the features of claim 1. Therefore paragraph 48 is completely irrelevant to the claimed feature, “determining whether space is available in an inode for a *file in the file system* and responsive to space being available, storing the *data* in the inode,” as recited in

claim 1.

Similarly, paragraph 53 discloses the use of an inode to store a data file. However, nothing in paragraph 53 or any other portion of *Crow* teaches the precondition to storing the data in the inode as recited in claim 1. In other words, *Crow* does not teach, “*determining* whether space is available in an inode *for a file in the file system* and *responsive* to space being available...,” as recited in claim 1. *Crow* does not perform a pre-determination prior to storing the data in the inode. *Crow* performs a post-determination. *Crow* performs the function of storing the data in the inode and “when the size of the data file surpasses the limited space available in the inode, the operating system converts the inode to an inode for storage of lists of extents.” *Crow*, paragraph 53. Therefore, this portion of *Crow* does not teach, “*determining* whether space is available in an inode *for a file in the file system* and *responsive* to space being available, storing the data in the inode,” as recited in claim 1.

Nevertheless, the Office Action now also refers to figures 9-11 of *Crow* as teaching the features of claim 1. Applicants address figure 10 of *Crow* vis-à-vis the rejection of claim 2. As shown below, figure 10 of *Crow* does not teach either the features of claim 1 or claim 2.

Additionally, Figures 9 and 11 also do not teach the features of claim 1. These figures are as follows:

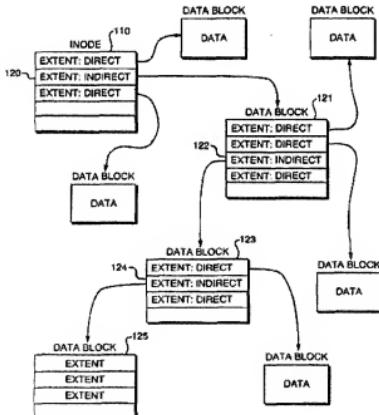


FIG. 9

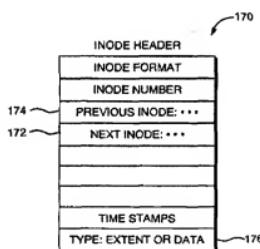


FIG. 11

Figure 9 of *Crow* shows the relationship among inode 110 and data blocks. Specifically, illustrates an example where the file system nests indirect extents. *Crow*, paragraph 0045. However, nothing in Figure 9 teaches, “*determining* whether space is available in an inode *for a file in the file*

system and responsive to space being available...,” as recited in claim 1.

Figure 11 of *Crow* shows details of an inode header. Specifically, figure 11 of *Crow* illustrates the headers 170 of one embodiment of the inodes 63, 64 of FIG. 5. *Crow*, paragraph 0050. Figure 11 of *Crow* shows that an inode header includes format, number, previous inode, next inode, time stamps, and the type - whether extent or data. However, nothing in Figure 1 teaches, “determining whether space is available in an inode for a file in the file system and responsive to space being available...,” as recited in claim 1.

As shown above, *Crow* does not teach all of the features of claim 1. Accordingly, *Crow* does not anticipate claim 1 or any other claim in this grouping of claims.

III.E.1.ii. *Rebuttal of the Office Action’s Response*

In response, *vis-à-vis* claim 1, the Office Action states that:

The Office Action respectfully disagreed with the Applicant’s argument above; since *Crow* discloses (paragraph [0044], [0047] and [0048]) “the operating system determines the maximum number of available contiguous blocks in each logical volume from data in the volume’s header or from information in a superblock spanning the entire storage system.”

Final Office Action of October 18, 2006, p. 5.

The Office Action makes a single counter argument based on the above-cited quotation. The Office Action does not address any of the specific points raised by Applicants.

However, the one quotation cited by the Office Action deals with the asserted feature in *Crow* that the operating system determines the maximum number of available contiguous blocks in each logical volume from data in the volume’s header, or from information in a superblock. However, this quotation has nothing to do with the claimed feature.

According to Applicant’s specification, p. 8, ll. 24-25, an inode is a data structure or record used to store information *about* files (emphasis supplied). This definition comports with what one of ordinary skill in the art knows an inode to be. Even if true, the fact that the operating system determines the maximum number of available contiguous blocks in each logical volume or from information in a superblock has little to do with the claimed feature of, “determining whether space is available in an inode for a file in the file system,” as in claim 1. Even if a superblock were somehow the equivalent of an inode, *Crow* would then only be teaching that the superblock contained information *about* files. However, *Crow* does not teach *determining* whether space is available *in* the inode for a file in the file system. Therefore, the Office Action has failed to rebut the fact that *Crow* does not teach all of the features of claim 1. Accordingly, *Crow* does not anticipate claim 1 or any other claim in this grouping of claims.

III.E.2. Claims 2, 3, 7, 10, 11, 15, 17, 18, and 22

III.E.2.i. *Response to Rejection*

Claim 2 is a representative claim of this grouping of claims. Claim 2 is as follows:

2. The method of claim 1 further comprising:
determining whether additional data is present; and
responsive to the additional data being present, storing the additional
data in a partially filled block of another file.

The Office Action rejects claim 2 as anticipated by *Crow*. Regarding claim 2, the Office Action asserts:

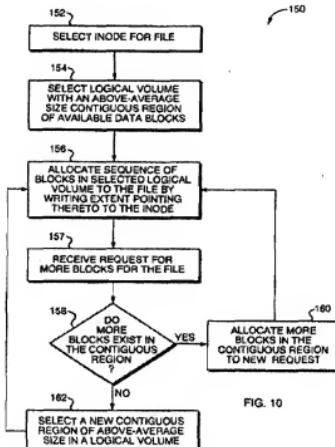
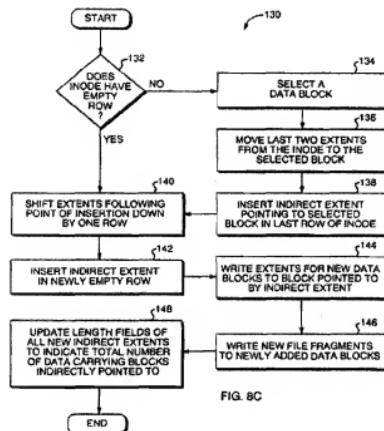
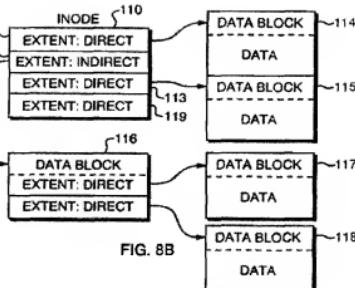
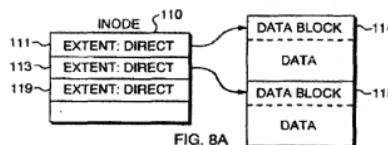
Crow discloses in figures 8A-8C1 and 10, to determining whether additional data
being present; and responsive to the additional data being present, storing the
additional data in a partially filled block of another file (paragraph [0038],
[0039], [0042] and [0044]).

Final Office Action dated October 18, 2006, p. 2.

Crow does not anticipate claim 2 because *Crow* does not teach all the features of claim 2. Claim 2 is as follows:

2. The method of claim 1 further comprising:
determining whether additional data is present; and
responsive to the additional data being present, storing the additional
data in a partially filled block of another file.

Because claim 2 depends from claim 1, the same distinctions between *Crow* and claim 1 apply to claim 2. Additionally, claim 2 claims other additional combinations of features not suggested by the reference. Specifically, *Crow* does not teach the feature of, “*storing the additional data in a partially filled block of another file*,” as recited in claim 2. The Office Action asserts otherwise, citing the following figures:



Figures 8A, 8B, and 8C do not teach the feature of, "storing the additional data in a partially filled block of another file." Instead, figures 8A and 8B show how the operating system uses indirect extents to grow the middle of a file. The indirect extent 112 points to more extents stored in a data block 116. These extents, in turn, point to *new* data block 117 and original data block 215. *Crow*, paragraph 38. This feature enables an operating system to logically insert a new data segment between any two selected data segments of a file without physically moving data blocks. *Crow*, paragraph 39. Figure 8C is a flow chart illustrating a method 130 of inserting a new file segment between two adjacent file segments. *Crow*, paragraph 39. In step 148, the operating system writes the new file segment in the *new* data block pointed to by the new direct extent. *Crow*, paragraph 44.

However, these figures do not teach “storing *the additional* data in a *partially filled block of another file*,” as recited in claim 2. Instead, these figures show how to add **new** data blocks to a file using *indirect extents*. Therefore, figures 8A, 8B, and 8C of *Crow* do not teach the claimed features of claim 2.

Furthermore, figure 10 of *Crow* does not teach the feature, “storing *the additional* data in a *partially filled block of another file*.” Figure 10 illustrates a method 150 of allocating data blocks to a file from a plurality of logical volumes. The operating system allocates a string of data blocks from the contiguous region of the selected volume to the file by writing an extent. The extent points to the string in the first row of the inode assigned to the file (step 156). Step 157 occurs when an application requests more data blocks for a file. If there are more data blocks contiguous to the physical location of the previous segment of the file, then the operating system allocates a new string of blocks immediately following the physical location of the previous segment. If no blocks contiguous to the previous segment are available, the operating system searches for a logical volume with a larger than average contiguous region of available data blocks (step 162).

However, nothing in figure 10 of *Crow* teaches the feature, “storing *the additional* data in a *partially filled block of another file*.” Instead, figure 10 of *Crow* teaches a method of storing data in contiguous blocks, which is different from the features of claim 2. Therefore, figure 10 of *Crow* does not teach the features of claim 2. Moreover, nothing in figure 10 of *Crow* teaches the features of claim 1.

Additionally, the Office Action cites to the following portions of *Crow* as disclosing the features of claim 2:

[0038] FIGS. 8A and 8B show how the operating system uses indirect extents to grow the middle of a file. FIG. 8A shows an inode 110 assigned to the file. The inode 110 has consecutive direct extents 111, 113, 119 that point to data blocks 114, 215, 330 storing originally consecutive segments of the file. FIG. 8B shows the final file in which an indirect extent 112 has been inserted between the two original direct extents 111, 119. The indirect extent 112 points to more extents stored in a data block 116. These extents, in turn, point to **new** data block 117 and original data block 215. Since the indirect extent 112 is physically located between the two original extents 111, 119, the segments stored in the blocks 117, 215 (indirectly pointed to) are logically located between the original segments stored in the blocks 114, 330. Inserting the indirect extent 112 has grown the middle of the associated file by logically inserting the segment in **new** data block 117 between the originally consecutive segments in data blocks 114 and 215.

Crow, p. 3, ¶ 38 (emphasis added).

[0039] The file system, illustrated in FIGS. 5-8B, allows any extent of an inode to be indirect, because the flag field indicates the type of each extent. This free placement of indirect extents within the inodes enables an operating system to logically insert a new data segment between any two selected data segments of a file without physically moving data blocks. To insert a new data segment, the system inserts an indirect extent into the file's inode between the two extents for the selected data segments. Then, the system makes the indirect extent point to a

data block storing new direct extents that point, in turn, to the consecutive pieces of new data segment. The new direct extents are logically located in the inode at the point where the new indirect extent has been inserted.

Crow, p. 3, ¶ 39 (emphasis added).

[0042] If the inode has an empty row, the operating system shifts down the original extents corresponding to segments that will follow the segments to be inserted by one row in the inode (step 134). Then the operating system inserts a new direct extent in the newly emptied row of the inode (step 136). Finally, the operating system writes the new file segment to a new data block pointed to by the new direct extent (step 138).

Crow, p. 3, ¶ 42 (emphasis added).

[0044] Next, the operating system inserts an indirect extent into the row of the inode previously occupied by the extent now in the second row of the indirect block (step 146). The new indirect extent points to the new indirect block and has a length equal to the sum of the lengths of both extents in the indirect block. In FIG. 8B, the operating system writes the extent 112 pointing to the data block 116 to the inode 110. Finally, the operating system writes the new file segment in the new data block pointed to by the new direct extent (step 148). In FIG. 8B, the new file segment is written to the data block 117.

Crow, p. 3, ¶ 44 (emphasis added).

None of the portions cited by the Office Action or any other portion of *Crow* teaches the feature of, “storing the additional data in a partially filled block of another file,” as recited in claim 2. Paragraphs 38 and 39 describe figures 8A and 8B. As previously discussed, this portion of *Crow* explains how to add new data blocks to a file using indirect extents. Similarly, paragraphs 42 and 44 describe figure 8C. Both paragraphs specifically state: “the operating system writes the new file segment in the new data block pointed to by the new direct extent.” Therefore, none of the portions cited by the Office Action or any other portion of *Crow* teaches the feature of, “storing the additional data in a partially filled block of another file,” as recited in claim 2. Accordingly, *Crow* does not anticipate claim 2 or any other claim in this grouping of claims.

III.E.2.ii. *Rebuttal of the Office Action’s Response*

In response to the above facts, the Office Action states that:

The Office Action respectfully disagreed with the Applicant’s argument above; since *Crow* discloses (paragraph [0036]) “a first portion of the flag field indicates whether the data blocks are locked or unlocked, that is, available or unavailable. The locked designation indicates that access to the data blocks is limited. The processors 44-45 and drivers 47-49 may change the flag field of an extent to the locked designation while manipulating data in the associated data blocks so that other devices do not access the data blocks in parallel. A second portion of the flag field indicates whether empty data blocks have been zeroed. By using the not

zeroed designation, the file system can allocate a data block to a file without zeroing the block beforehand. If a subsequent access writes the entire data block, the block will not have to be zeroed saving processing time. A third portion of the flag field categorizes the data type stored in a data block into one of three types, that is, real file data, non-data, or extents".

Final Office Action dated October 18, 2006, p. 6.

The Office Action accurately quotes paragraph 36 of *Crow*, but misinterprets the meaning of paragraph 36 *vis-à-vis* the features of claim 2. The relevant portion of the above-quoted text is as follows:

A second portion of the flag field indicates whether *empty data blocks have been zeroed*. By using the not zeroed designation, the file system can allocate a data block to a file without zeroing the block beforehand.

Crow, selected portion of paragraph 36 (emphasis supplied).

Thus, *Crow* states that the second flag field of an inode extent indicates whether an *empty* data block has been zeroed. Thus, the data block is already empty, regardless of whether the data block has been zeroed. The term zeroed therefore applies to a concept *other than whether the data block is partially filled*, as required by claim 2. *Crow* explicitly provides that regardless of whether the data block is "zeroed," the data block is empty. Thus, *Crow* does not teach the claimed feature of, "responsive to the additional data being present, storing the additional data in a partially filled block of another file," as in claim 2. Accordingly, *Crow* does not anticipate claim 2 or any other claim in this grouping of claims.

III.E.3. Claims 4, 12, and 19

Claim 4 is a representative claim of this grouping of claims. Claim 4 is as follows:

4. The method of claim 3, wherein the partially filled block is a last block of the another file.

The Office Action rejects claim 4 as anticipated by *Crow*. Regarding claim 4, the Office Action states that:

Crow discloses in figure 8C, wherein the partially filled block being a last block of the another file (paragraph [0042]).

Final Office Action dated October 18, 2006, p. 3.

The portion of *Crow* cited by the Office Action is as follows:

[0042] If the inode has an empty row, the operating system shifts down the original extents corresponding to segments that will follow the segments to be inserted by one row in the inode (step 134). Then the operating system inserts a new direct extent in the newly emptied row of the inode (step 136). Finally, the operating system writes the new file segment to a new data block pointed to by the new direct extent (step 138).

Crow, paragraph 0042.

The cited portion of *Crow* teaches that if the inode has an empty row, the operating system shifts original extents in the inode by one row. The operating system then inserts a new direct extent into the newly emptied row of the inode. The operating system then writes the new file segment to a new data block. (Applicants note that the file segment is written to the data block to which the extent points, and not to the inode itself).

However, *Crow* in no way teaches that, “the partially filled block is a *last* block of the another file.” In fact, *Crow* does not teach partially filled blocks at all, so *Crow* cannot teach this claimed feature. Certainly, *Crow* does not mention that a partially filled block is a *last* block of the another file, as claimed. Accordingly, *Crow* does not anticipate claim 4 or any other claim in this grouping of claims.

III.E.4. Claims 5, 13, and 20

Claim 5 is a representative claim of this grouping of claims. Claim 5 is as follows:

5. The method of claim 1, wherein the space is located in an extension area in the inode.

The Office Action rejects claim 4 as anticipated by *Crow*. Regarding claim 5, the Office Action states that:

Crow discloses in figures 5-10, wherein the space being located in an extension area in the inode.

Final Office Action dated October 18, 2006, p. 3.

None of Figures 5-10 teach the claimed feature that, “the space is located in an extension area in the inode.” The Office Action does not point to any particular part of these figures as teaching this claimed feature. Instead of showing that each individual figure fails to show the features of claim 4, Applicants use Figure 7 of *Crow* to prove that the opposite is true; namely, that *figures in Crow specifically do not teach* that the space is located in an extension area in the inode, as claimed. A similar analysis applies to each of Figures 5, 6, and 8-10. Figure 7 of *Crow* is as follows:

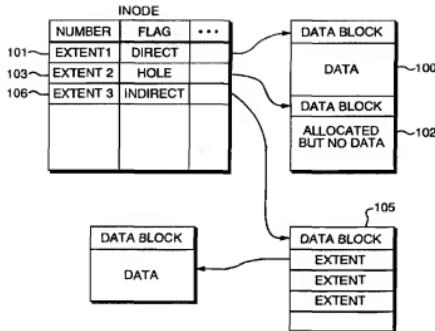


FIG. 7

Figure 7 of *Crow* shows that extents in an inode *point to* data blocks. The *data blocks* contain the stored data. The inode extents do not store the data, only the data blocks store data. Therefore, these figures do not teach the claimed feature of, “wherein the space is located in an *extension area* in the inode.” Accordingly, *Crow* does not anticipate claim 5 or any other claim in this grouping of claims.

The portion of *Crow* closest to the features of claim 5 is as follows:

[0053] The operating system writes the binary value to the third entry 176 to indicate storage of a data file when the associated inode is first created. Then, the operating system uses the inode to store the associated data file. When the size of the data file surpasses the limited space available in the inode, the operating system converts the inode to an inode for storage of lists of extents.

Crow, paragraph 53.

This portion of *Crow* states that the operating system uses the inode to store the associated data file. When the size of the data file surpasses the space in the inode, the operating system converts the inode to an inode for storage of lists of extents.

However, again *Crow* does not teach that the data is stored in the *extents themselves*, as required by claim 5. Therefore, this portion of *Crow* does not teach all of the features of claim 5. Moreover, no portion of *Crow* teaches these claimed features. Accordingly, *Crow* does not anticipate claim 5 or any other claim in this grouping of claims.

III.E.5. Claims 6, 14, and 21

Claim 6 is a representative claim of this grouping of claims. Claim 6 is as follows:

6. The method of claim 1 further comprising:

determining whether a file size for the data is divisible by a block size for blocks in the file system; and if the file size is divisible by the block size, storing the data in a block.

The Office Action rejects claim 6 as anticipated by *Crow*. Regarding claim 6, the Office Action states that:

Crow discloses further comprising determining whether a file size for the data being divisible by a block size for blocks in the file system; and if the file size is divisible by the block size, storing the data in a block (paragraph [0031], [0034]).

Final Office Action dated October 18, 2006, p. 3.

The first portion of *Crow* cited by the Office Action is as follows:

[0031] Each data block 80-82, 84-85, 92-94 has the same size, for example, 4K bytes. Nevertheless, the extents 65-66 can map file segments of different sizes to physical storage locations. To handle file segments of different sizes, each extent has a length field that indicates the number of data blocks in the string of data blocks that stores the associated file segment.

Crow, paragraph 0031.

This portion of *Crow* teaches that data blocks have the same size. Extents in an inode can map file segments of different sizes to different physical storage locations. An extent length field in the inode indicates the number of data blocks in the string of data blocks that stores the associated file segment.

However, *Crow* does not teach *determining* whether a file size is divisible by a block size for blocks in the file system. *Crow* does not actually store data in a block *if* the file size is divisible by the block size, as claimed. Instead, *Crow* only teaches that large files are stored by appending small data blocks. Because this feature is not equivalent to the claimed feature, this portion of *Crow* does not teach the features of claim 6.

Nevertheless, the Office Action quotes from a second portion of *Crow* as teaching the features of claim 6. The second portion of *Crow* cited by the Office Action is as follows:

[0034] The address pointer field indicates both a logical volume and a physical offset of a data block in the logical volume. In one embodiment, the pointer fields for the logical volume and the data block therein are 2 bytes and 4 bytes long, respectively. For this field size and data blocks of 32 kilobytes, the extent fields can identify about 140.times.10.sup.12 bytes of data in each of about 64K different logical volumes. Thus, the file system of the distributed storage system 40 can handle very large files.

Crow, paragraph 0034.

This portion of *Crow* teaches that the address pointer field indicates both a logical volume and a physical offset of a data block in the logical volume. *Crow* also describes how the described system can handle very large files using small data blocks. However, again, *Crow* does not teach *determining*

whether a file size is divisible by a block size for blocks in the file system. *Crow* does not actually store data in a block *if* the file size is divisible by the block size, as claimed. Instead, *Crow* only teaches that large files are stored by appending small data blocks among different logical and physical volumes. Because this feature is not equivalent to the claimed feature, this portion of *Crow* does not teach the features of claim 6.

Neither of the portions of *Crow* teach all of the features of claim 6. Therefore, *Crow* does not anticipate claim 6 or any other claim in this grouping of claims.

III.E.6. *Crow* Generally

Applicants have thoroughly rebutted the Office Action's assertion that *Crow* anticipates the claims. As shown above, *Crow* fails to teach the *responsive to* and *determining* features of claim 1, as well as the other portions of claim 1 pointed-out above. Therefore, under the standards of *In re Bond*, *Crow* does not anticipate claim 1.

IV. Conclusion

The subject application is patentable over the cited references and should now be in condition for allowance. The Office Action is invited to call the undersigned at the below-listed telephone number if in the opinion of the Office Action such a telephone conference would expedite or aid the prosecution and examination of this application.

DATE: October 16, 2007

Respectfully submitted,

/Theodore D. Fay III/

Theodore D. Fay III
Reg. No. 48,504
Yee & Associates, P.C.
P.O. Box 802333
Dallas, TX 75380
(972) 385-8777
Attorney for Applicants

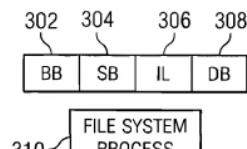
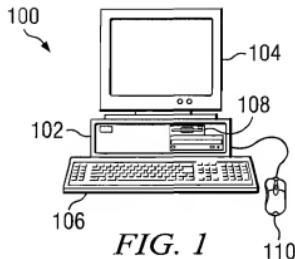


FIG. 3

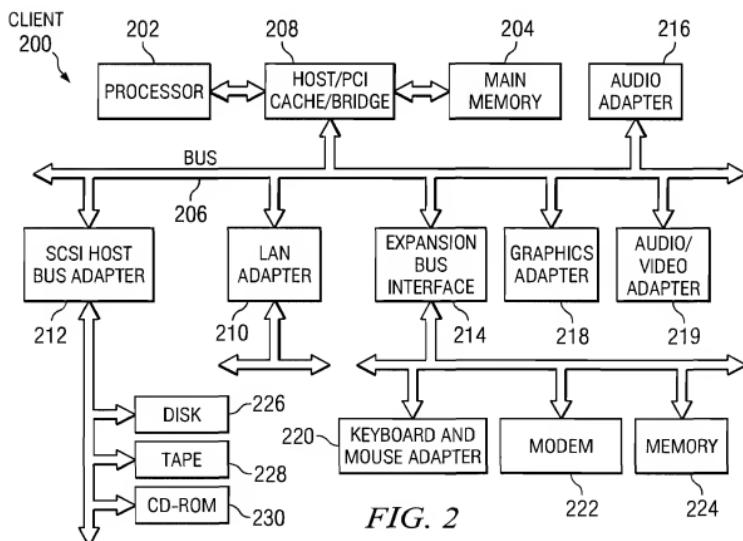


FIG. 2